Generate Faces

REVIEW | CODE REVIEW | HISTORY

Meets Specifications

Your project meets all the requirements on the first submission, well done! 👍

You showed a good understanding of the concepts of this project and implemented all the functions perfectly.

We are looking forward to your next submissions in the nanodegree! :-)

Except for more data and epoch, how I could actually improve my model?

For the discriminator and generator you could try the following:
1) Use custom weight initialization (e.g. Xavier weight initialization) to help converge faster by breaking symmetry or you can also use truncated_normal_initializer with stddev=0.02, which improve overall generated image quality, like in DCGAN paper.

2) Experiment with various values of negative slopes (slope of the leaky Relu as stated in DCGAN paper) between 0.06 and 0.18 and compare your results.

3) Experiment with dropout layers for discriminator. If discriminator end up dominating generator, we must reduce discriminator learning rate and increase dropout. (CONV/FC -> BatchNorm -> ReLu(or other activation) -> Dropout -> CONV/FC)

For the discriminator loss: use label smoothing, so that the discriminator doesn't get too strong and to generalize in a better way. See also https://arxiv.org/abs/1606.03498.

For the hyperparameters:
1) Try using different values of learning rate between 0.0002 and 0.0008, DCGAN remains stable with values in this range.

2) Experiment with different values of beta1 between 0.2 and 0.5 and compare your results.

3) An important point to note is, batch size and learning rate are linked. If the batch size is too small then the gradients will become more unstable and would need to reduce the learning rate and vice versa. Choose values between 16 and 32.

Further reading:
https://github.com/soumith/ganhacks#how-to-train-a-gan-tips-and-tricks-to-make-gans-work
http://blog.otoro.net/2016/04/01/generating-large-images-from-latent-vectors/

Required Files and Tests

The project submission contains the project notebook, called “dlnd_face_generation.ipynb”.

All required files are included 👌

All the unit tests in project have passed.

Excellent!

Data Loading and Processing

The function `get_dataloader` should transform image data into resized, Tensor image types and return a DataLoader that batches all the training data into an appropriate size.

The image are correctly resized and transformed to a Pytorch tensor 👌

Pre-process the images by creating a `scale` function that scales images into a given pixel range. This function should be used later, in the training loop.

Pixels values values are scaled in the right way!

Build the Adversarial Networks

The Discriminator class is implemented correctly; it outputs one value that will determine whether an image is real or fake.

The `Discriminator` class is correctly implemented, you have implemented multiple convolutional layers with batch norm and leaky relu activation function and returned one logit value.

The Generator class is implemented correctly; it outputs an image of the same shape as the processed training data.

Also the `Generator` class is correctly implemented! 👌

This function should initialize the weights of any convolutional or linear layer with weights taken from a normal distribution with a mean = 0 and standard deviation = 0.02.

Good work! The weights are initialized in a similar way as the original DCGAN paper.

Optimization Strategy

The loss functions take in the outputs from a discriminator and return the real or fake loss.

The loss functions are correctly implemented, well done!

There are optimizers for updating the weights of the discriminator and generator. These optimizers should have appropriate hyperparameters.

Optimizers are defined correctly 👍

Training and Results

Real training images should be scaled appropriately. The training loop should alternate between training the discriminator and generator networks.

Perfect implementation of the training code using the previously defined functions!

The loss functions converge around a fixed value which is a good sign.

There is not an exact answer here, but the models should be deep enough to recognize facial features and the optimizers should have parameters that help wth model convergence.

The project generates realistic faces. It should be obvious that generated sample images look like faces.

Your generated faces look almost as good as the original ones, great work!

The question about model improvement is answered.

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review